Brief Announcement: Hierarchical Consensus

Benjamin Bengfort and Pete Keleher
{*bengfort,keleher*}*@cs.umd.edu*}

1 Introduction

We introduce *Hierarchical Consensus*, an approach to generalizing consensus that allows us to scale groups beyond a handful of nodes, across wide areas. Hierarchical Consensus (HC) increases the availability of consensus groups by partitioning the decision space and nominating distinct leaders for each partition. Partitions eliminate distance by allowing decisions to be co-located with replicas that are responding to accesses. Hierarchical consensus is flexible locally, but improves upon prior approaches [4, 2, 1, 5, 6, 3] by balancing load, allowing fast replication across wide areas, and enabling consensus across large (>100) systems of devices.

Our default use case is in maintaining *linearizablity* across `read` and `update` operations used to support a wide-area object store or file system. We consider a set of processes $P = \{p_i\}_{i=1}^{n}$ which are connected via an asynchronous network, whose connections are highly variable. The variability of a communication link between p_i and p_j is modulated by the physical distance of the link across the geographic wide area. Each process maintains the state of a set of objects, $O = \{o_i^v\}_{i=1}^{m}$, which are accessed singly or in groups at a given process and whose state is represented by a monotonically increasing version number, v.

2 Hierarchical Consensus

Hierarchical consensus conducts coordination decisions as a tier of quorums such that parent quorums manage the *decision space* and leaf quorums manage *access ordering.* Hierarchical consensus considers the decision space as subsets of the object set, and subquorums are defined by a time-annotated disjoint subset of the objects they maintain, $Q_{i,e} \subset O$. The set of subquorums, Q, is not a complete partition of O, but only represents the set of objects that are being accessed at time e.

The hierarchical consensus algorithm starts with a root quorum whose primary responsibilities are i) the mapping of objects to subquorums and ii) the mapping of replicas to subquorums. Each instance of such a map defines a distinct *epoch*, e, a monotonically increasing representation of the term of Q_e. Decisions that require a change of the decision space or changes the mapping of objects or replicas to subquorums requires a new *epoch.* The fundamental relationship between epochs is as follows: any access that happens in epoch $e \rightarrow e + 1$. Alternatively, any access in epoch $e + 1$ *depends on* all accesses in epoch e.

All accesses to an object must be forwarded to the leader of the subquorum that contains the object. Objects that are accessed together or who have application-specific, explicit dependencies (such as the set of objects included in a transaction) must be part of the same subquorum so that local accesses are totally ordered. Dependent objects that are not part of the same subquorum require either a change in epoch or a mechanism to allow *remote accesses*, which we will discuss in a following section. Accesses in different subquorums but in the same epoch happen concurrently from the global perspective (but are non-conflicting), though accesses in a specific subquorum are totally ordered locally.

2.1 Operation

The root consensus group coordinates all decision space changes. Consider the simple example of the transfer of object responsibility from one subquorum to another:

$Q_1 : \{o_a, o_b, o_c\} \rightarrowtail \{o_a, o_b, o_g, o_h\}$
$Q_2 : \{o_d, o_e, o_f, o_g, o_h\} \rightarrowtail \{o_c, o_d, o_e, o_f\}$

Each of the two subquorums, Q_1 and Q_2, wants to give up a portion of its existing decision space and to add objects currently mapped to another subquorum. Reallocating subquorums requires a two phase consensus decision. Both subquorum leaders send change requests to the leader of the parent quorum, which may aggregate several requests into a single namespace change. While the parent quorum gets consensus to make the epoch change, subquorums can continue operating on their own decision space. Once the parent quorum updates the epoch, it communicates the change to all affected subquorums. Each subquorum independently decides when to transition to the new epoch. Subquorums in the new epoch can only access newly-gained objects once they have been released by the objects' owners in the last epoch.

2.2 Epochs and Ordering

Hierarchical consensus requires all accesses in each subquorums to be linearizable, guaranteed by serializing all accesses through the subquorum leader. Global linearizability is guaranteed by serializing epochs at the parent quorum, and limiting clients to access only one subquorum per epoch (relaxed in Section 2.2.1).

Let *interval* i_e be the ordered set of accesses of the replicas in subquorum Q_i during epoch e. We enforce linearizable ordering of all accesses in the entire system by ensuring that there must exist a total ordering of the intervals that produces the correct access results. Access results should be equivalent to any interval ordering such that all intervals in e occur before intervals in $e+1$ (our "interval ordering" invariant). This is because there is no cross-traffic between any Q_i and Q_j, and therefore ordering interval i_e before j_e is exactly the same as ordering interval j_e before i_e, for any i, j, and e.

The internal invariant requires $\forall_{x,y} : Q_{x,e} \rightarrow Q_{y,e+1}$. Ordering all accesses according to consensus-based log order and interval order satisfies both the internal invariant and linearizability while still allowing subquorums to operate independently within epochs. Given Q_i and Q_j within epochs $e = 1$ and $e = 2$, one possible interval order is $Q_{i,1} \rightarrow Q_{j,1} \rightarrow Q_{i,2} \rightarrow Q_{j,2}$.

2.2.1 Remote Accesses

By default we assume that the set of replicas *assigned* to subquorums are also disjoint, and that all accesses through a replica of a given subquorum are mapped to the local decision space. This is often reasonable. However, if an object is assigned to a decision space and a replica in another subquorum wishes to access it, the system must either disallow the access (our default approach) or take explicit notice that a dependency has been created between the subquorums.

The latter approach requires a serialization of all accesses currently within the remote quorum with respect to all accesses before the remote access in the local quorum. Assume a read access from Q_k to Q_i in epoch e; at the time of the read all accesses in Q_i must $\rightarrow$ all accesses following the read. We accommodate this requirement by using the read endpoints to break interval i_e into $i_{e.1}$ and $i_{e.2}$ at the point Q_i receives the remote access, and interval k_e into $k_{e.1}$ and $k_{e.2}$ at the point Q_k receives a response as shown in Figure 1. Our results are consistent with total interval ordering by incorporating $Q_{i.1} \rightarrow Q_{i.2}$. Remote accesses are expensive and the runtime system must weigh the cost of repeated remote accesses against the cost of epoch changes.

2.2.2 Fuzzy Epochs

Only subquorums involved in decision space changes need take notice of epoch changes. Other subquorums can move to a new epoch at no cost when informed of new epoch numbers from remote requests. Slow-responding subquorums therefore do not block decision space changes for

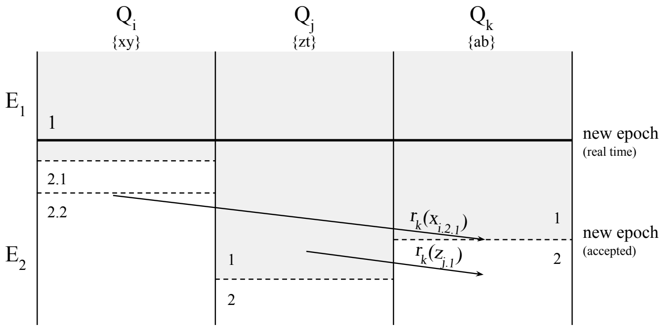

Figure 1: The gray region shows the "fuzzy" boundary between epochs E_1 and E_2. The first read access, $r_k(x_{2.1})$, reads the value of object x from Q_i into Q_k and forces Q_k to change epochs.

other quorums. Safety is guaranteed because no writes in the next epoch depend on these writes. These "fuzzy epochs" (Figure 1) allow an epoch change to be implemented solely by the root quorum, allowing subquorums to move to the new epoch independently. This flexibility is key to coping with partitions and varying connectivity in the wide area.

3 Correctness Sketch

We assert that consensus at the leaf nodes is correct and safe because decisions are implemented using well-known leader-oriented consensus approaches. Hierarchical consensus therefore has to demonstrate linearizable correctness and safety between subquorums for a single epoch and between epochs. Briefly, linearizability requires that external observers view operations to objects as instantaneous events. Within an epoch, the subquorum coordinates read and write accesses, and thus guarantees linearizability for all replicas in that quorum. Remote accesses and the internal invariant also enforce linearizability of accesses between subquorums. Epoch transitions raise the possibility of objects being re-assigned from one subquorum to another, with each subquorum making the transition independently. Correctness is guaranteed by an invariant that acquiring subquorums delay accessing newly acquired objects until receiving notice that the releasing subquorum(s) have transitioned, *plus* a description of object versions at the point of this transition.

References

[1] Martin Biely, Zoran Milosevic, Nuno Santos, and Andre Schiper. S-paxos: Offloading the leader for high throughput state machine replication. In *Reliable Distributed Systems (SRDS), 2012 IEEE 31st Symposium on*, pages 111–120. IEEE, 2012.

[2] H. Howard, D. Malkhi, and A. Spiegelman. Flexible Paxos: Quorum intersection revisited. *ArXiv e-prints*, August 2016.

[3] Tim Kraska, Gene Pang, Michael J. Franklin, Samuel Madden, and Alan Fekete. MDCC: Multi-data center consistency. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 113–126. ACM, 2013.

[4] Leslie Lamport. Paxos made simple. *ACM Sigact News*, 32(4):18–25, 2001.

[5] Yanhua Mao, Flavio Paiva Junqueira, and Keith Marzullo. Mencius: Building efficient replicated state machines for WANs. In *OSDI*, volume 8, pages 369–384, 2008.

[6] Iulian Moraru, David G. Andersen, and Michael Kaminsky. There is more consensus in egalitarian parliaments. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 358–372. ACM, 2013.